10/002480

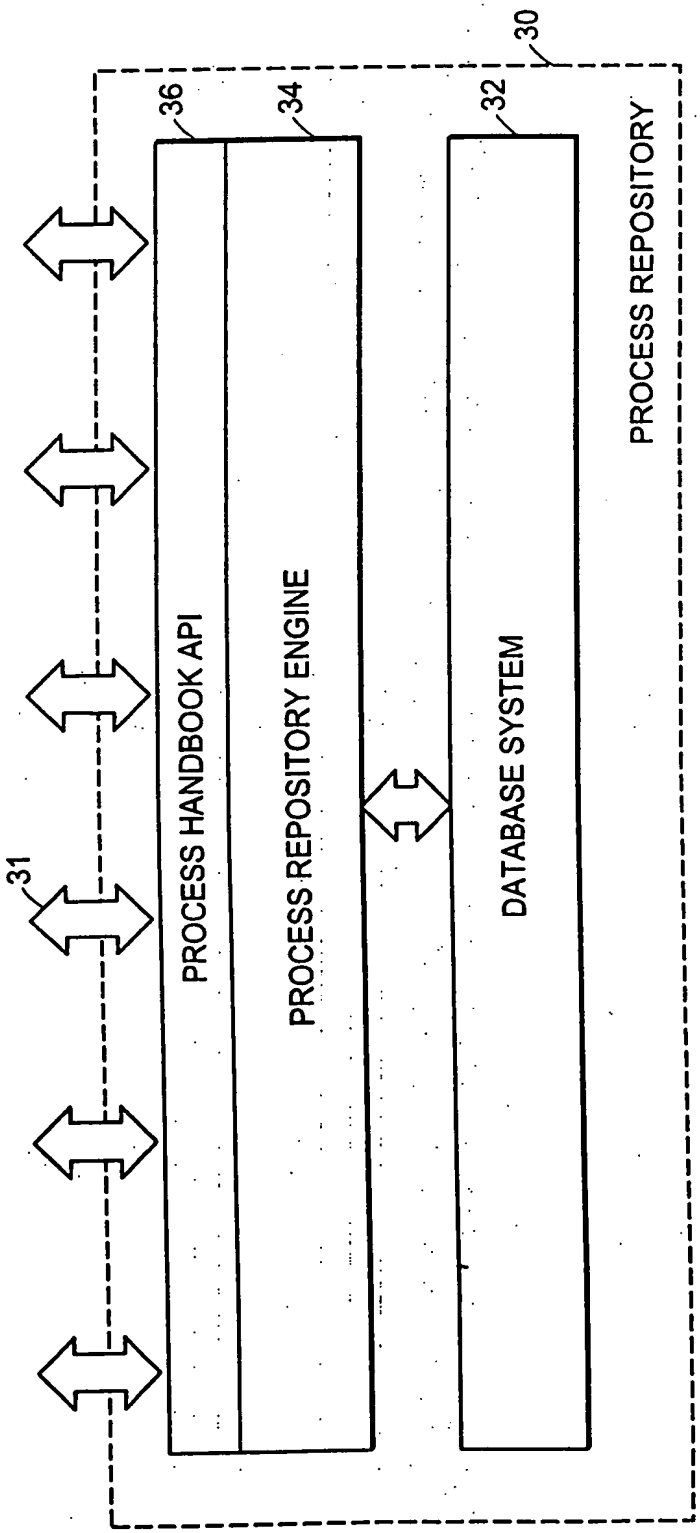


FIG. 1

20040804.0800

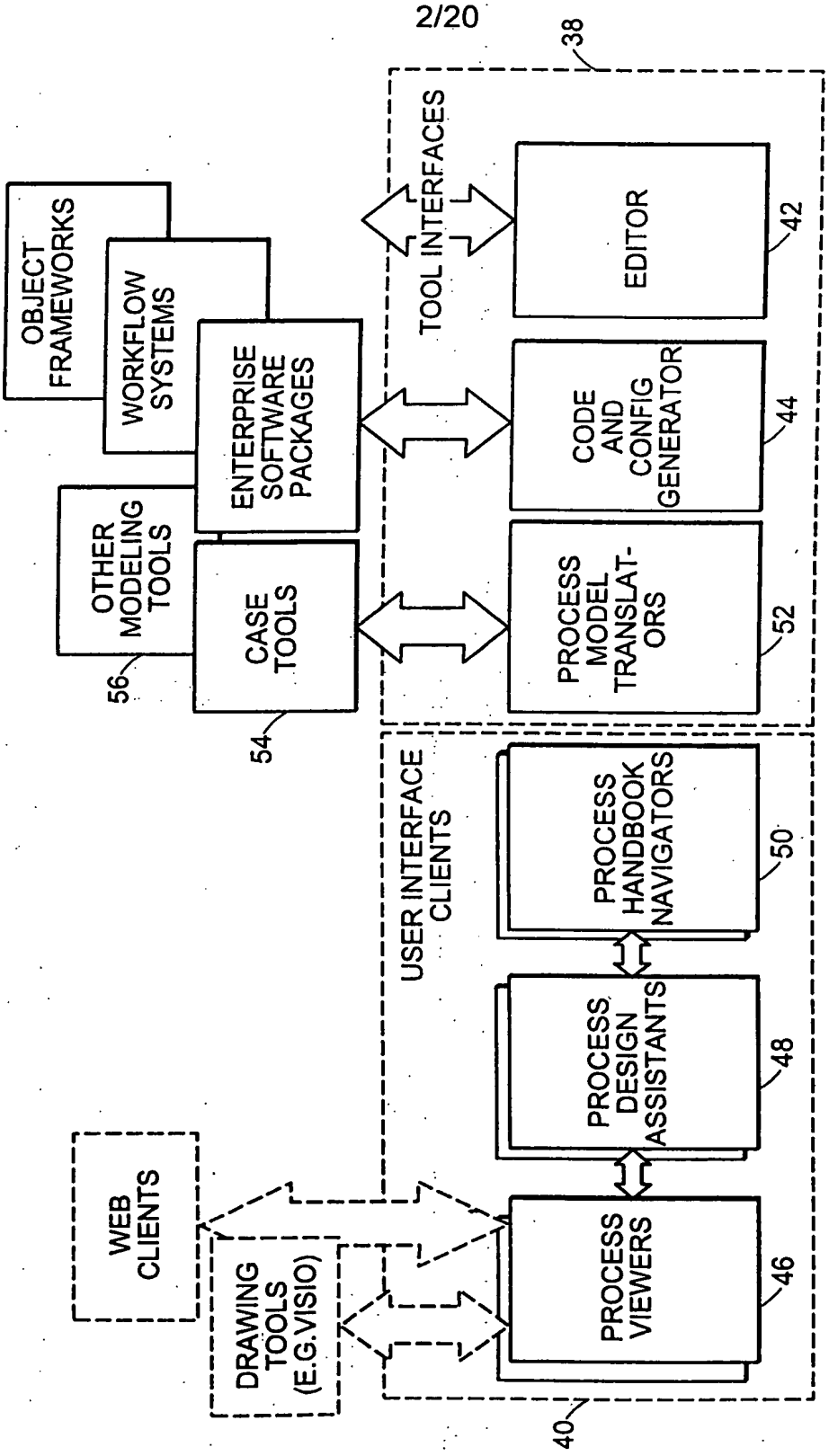

38
TOOL INTERFACES
EDITOR
42
CODE AND CONFIG GENERATOR
44
PROCESS MODEL TRANSLAT-ORS
52
OBJECT FRAMEWORKS
WORKFLOW SYSTEMS
ENTERPRISE SOFTWARE PACKAGES
OTHER MODELING TOOLS
CASE TOOLS
56
54
USER INTERFACE CLIENTS
PROCESS HANDBOOK NAVIGATORS
50
PROCESS DESIGN ASSISTANTS
48
PROCESS VIEWERS
46
WEB CLIENTS
DRAWING TOOLS (E.G.VISIO)
40

FIG. 2

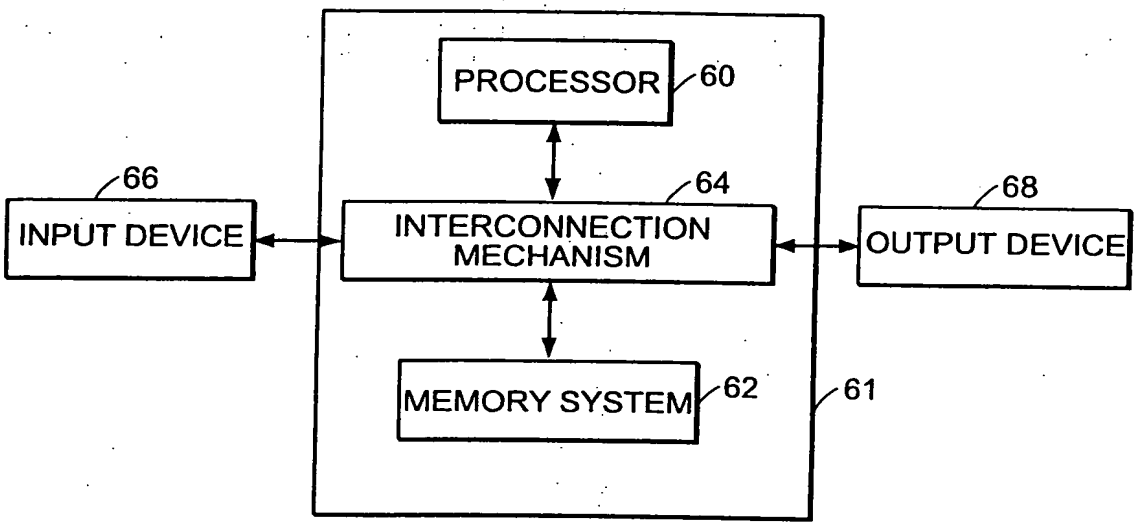


FIG. 3

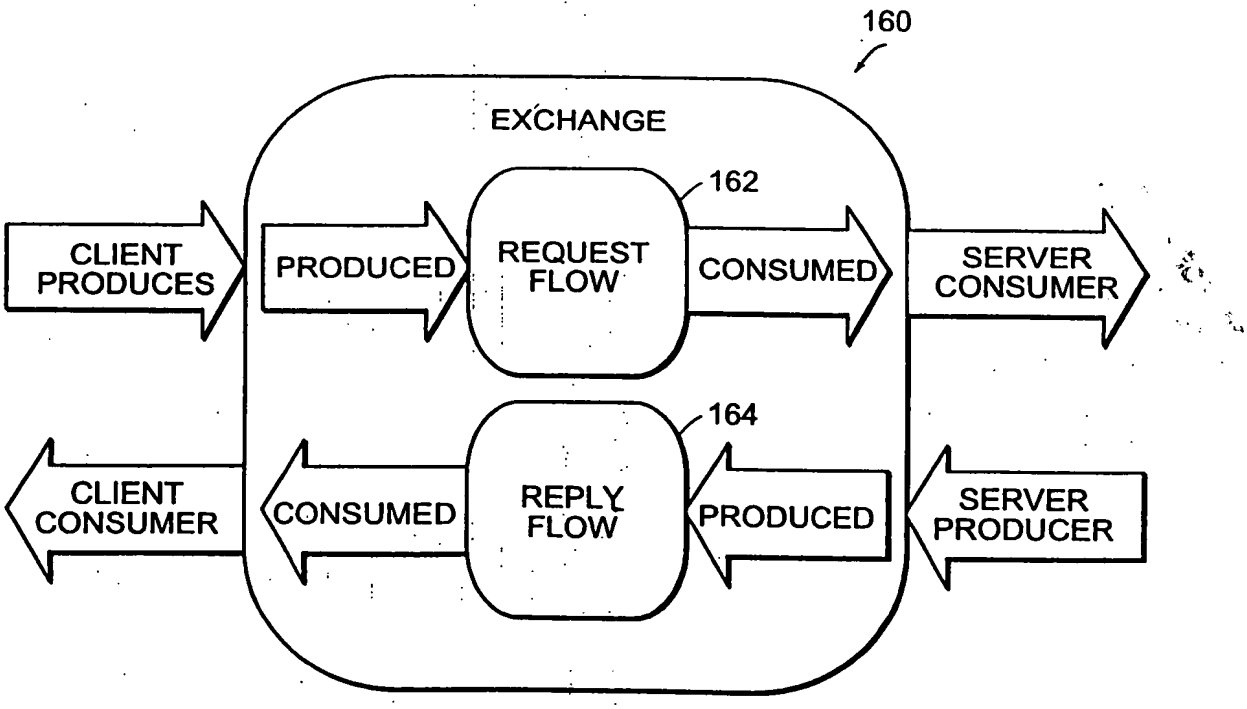


FIG. 5

20040808007

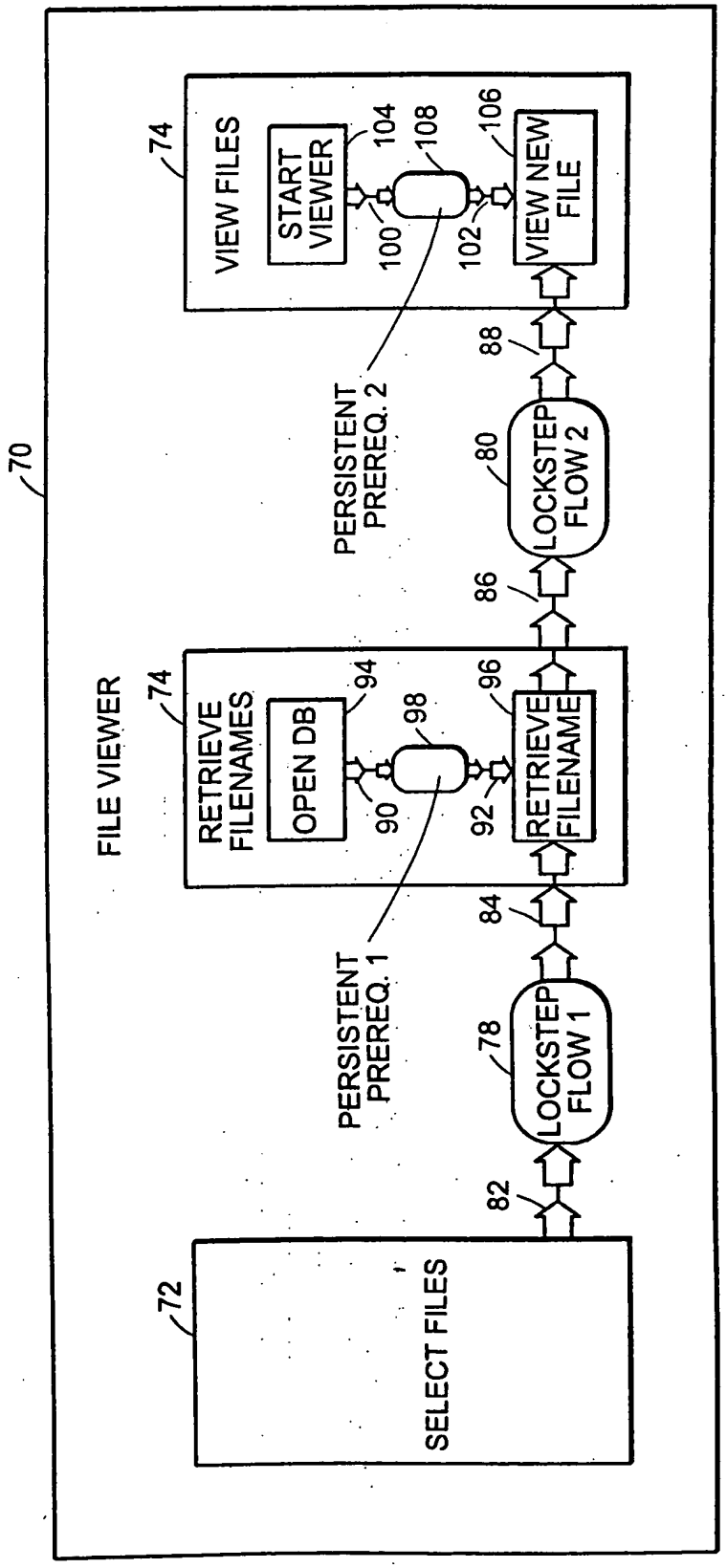


FIG. 4

20040804.0800

20040902.040040.0002

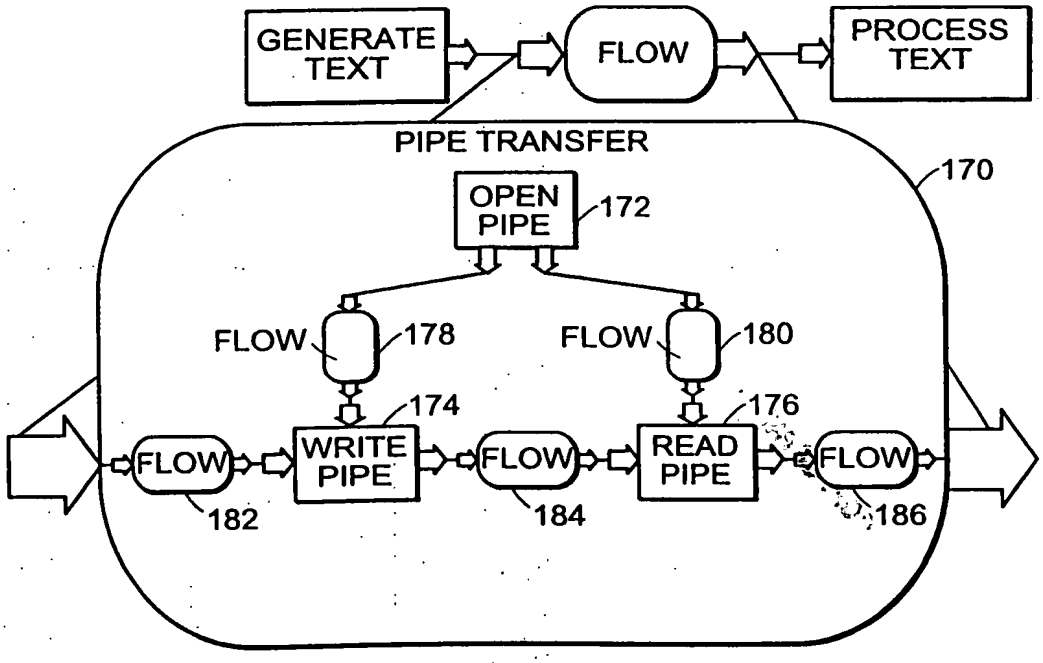


FIG. 6

FIG. 7A

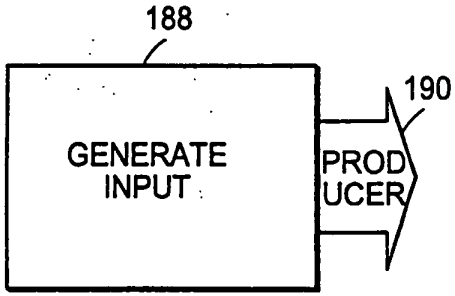


FIG. 7B

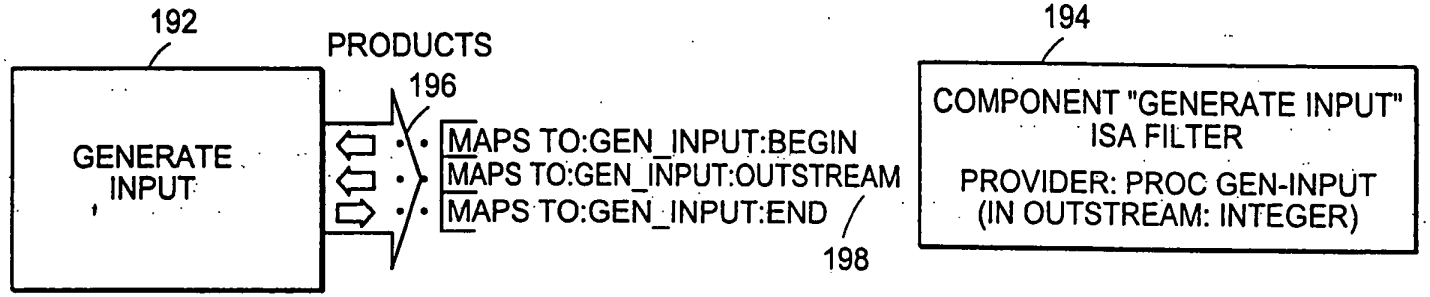
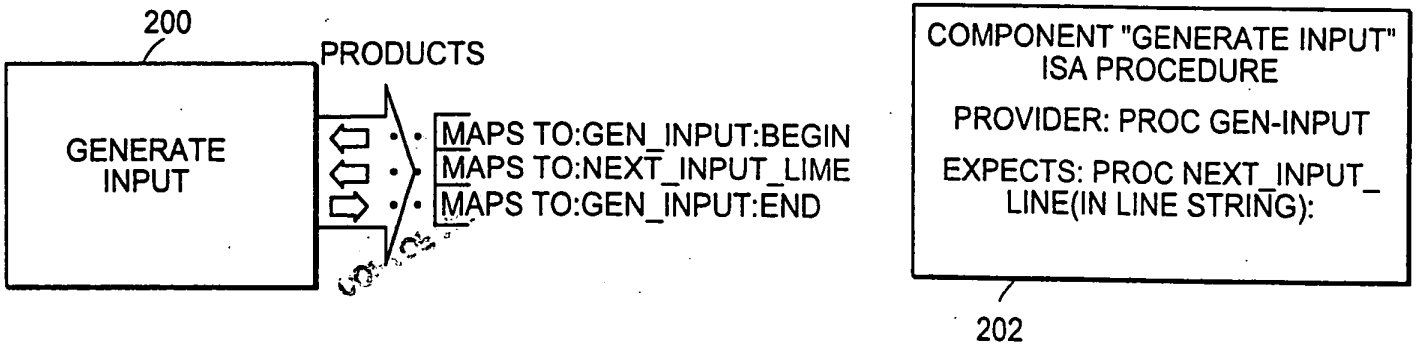


FIG. 7C



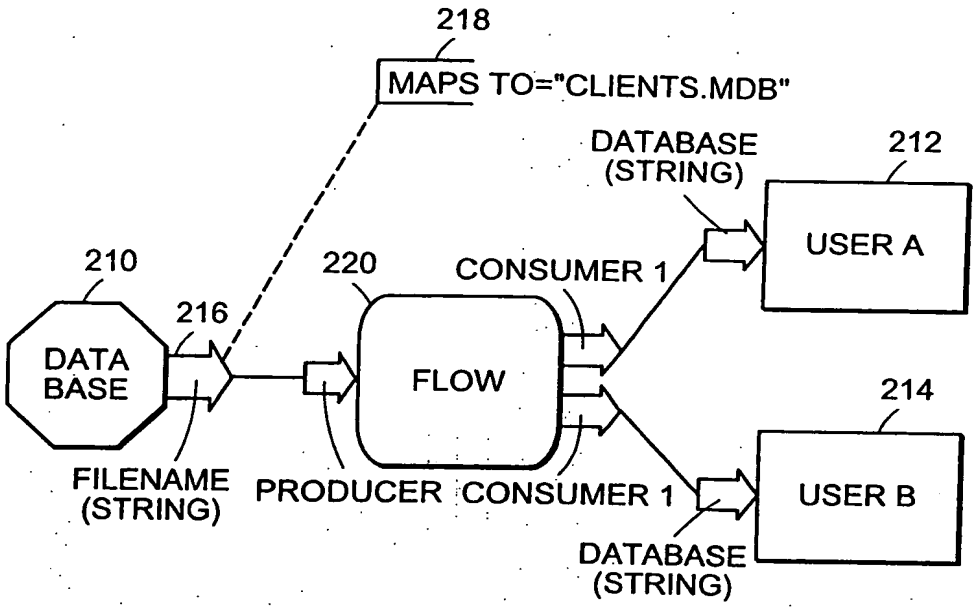

218
MAPS TO="CLIENTS.MDB"
DATABASE (STRING)
212
USER A
210
220
CONSUMER 1
216
DATA BASE
FLOW
214
FILENAME (STRING)
PRODUCER
CONSUMER 1
USER B
DATABASE (STRING)

FIG. 8

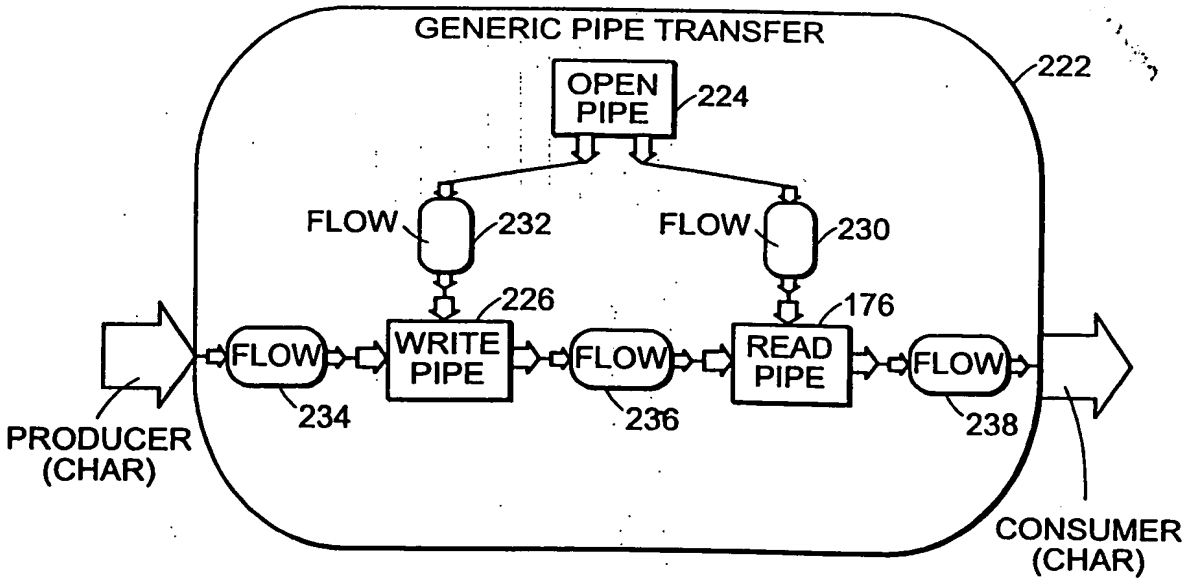

GENERIC PIPE TRANSFER
222
OPEN PIPE
224
FLOW
232
FLOW
230
226
176
FLOW
WRITE PIPE
FLOW
READ PIPE
FLOW
234
236
238
PRODUCER (CHAR)
CONSUMER (CHAR)

FIG. 9

PH_Object - 110

Unique Id	112
Name	114
Kind	116
Creator, etc.	118

FIG. 10

Parents -120

Object Id	122
Parent Id	124

FIG. 11

Decomposition - 130

Owner Id	132
Slot Id	134
Slot Value	136
Kind	138
Additional info...	140

FIG. 12

Connectors - 142

Connector Id	144
Owner Id	146
Endpoint 1	148
Endpoint 2	150

FIG. 13

20040204208

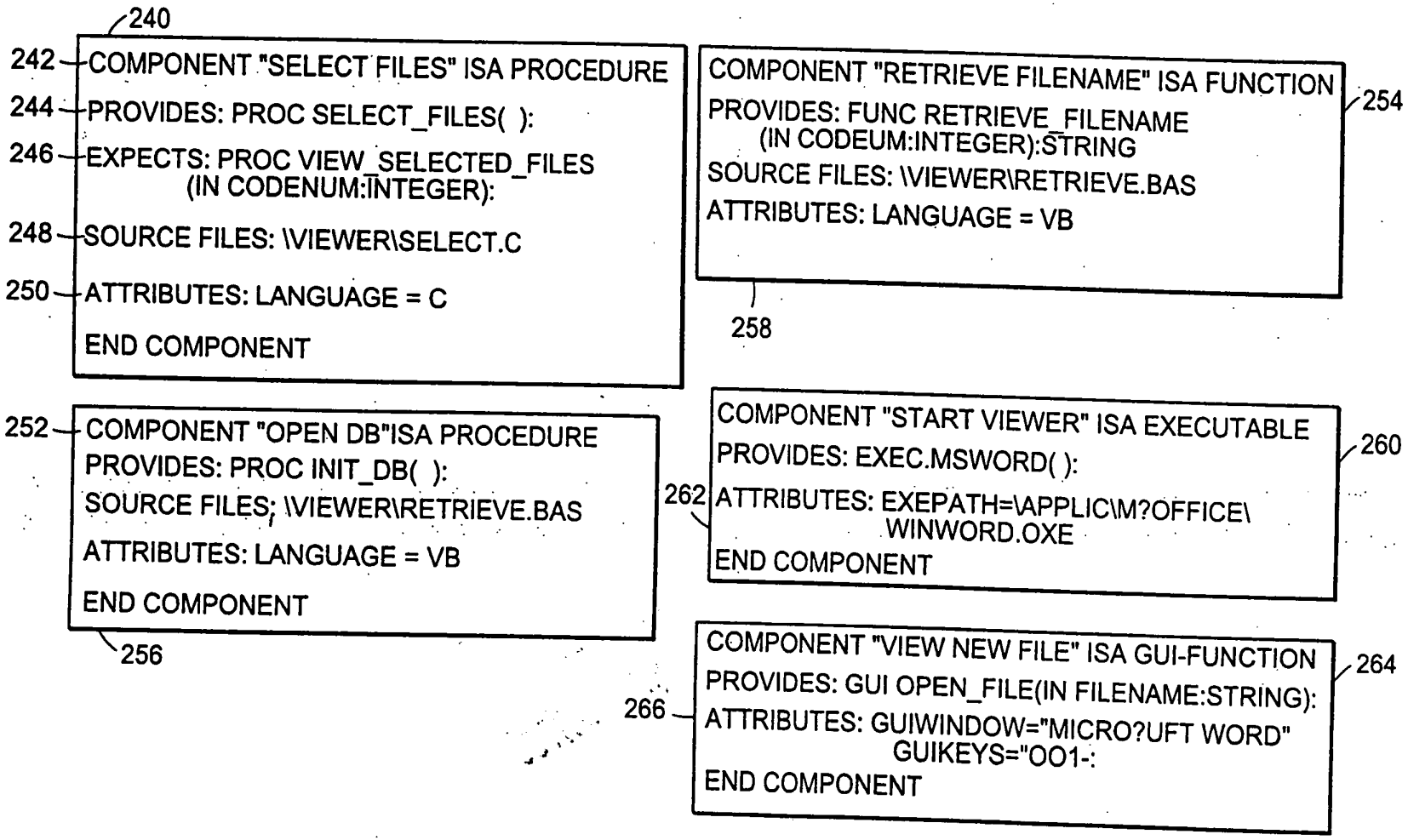

240
242
COMPONENT "SELECT FILES" ISA PROCEDURE
244
PROVIDES: PROC SELECT_FILES( ):
246
EXPECTS: PROC VIEW_SELECTED_FILES
(IN CODENUM:INTEGER):
248
SOURCE FILES: \VIEWER\SELECT.C
250
ATTRIBUTES: LANGUAGE = C
END COMPONENT
COMPONENT "RETRIEVE FILENAME" ISA FUNCTION
254
PROVIDES: FUNC RETRIEVE_FILENAME
(IN CODEUM:INTEGER):STRING
SOURCE FILES: \VIEWER\RETRIEVE.BAS
ATTRIBUTES: LANGUAGE = VB
258
252
COMPONENT "OPEN DB"ISA PROCEDURE
PROVIDES: PROC INIT_DB( ):
SOURCE FILES; \VIEWER\RETRIEVE.BAS
ATTRIBUTES: LANGUAGE = VB
END COMPONENT
256
COMPONENT "START VIEWER" ISA EXECUTABLE
260
262
PROVIDES: EXEC.MSWORD( ):
ATTRIBUTES: EXEPATH=\APPLIC\M?OFFICE\
WINWORD.OXE
END COMPONENT
COMPONENT "VIEW NEW FILE" ISA GUI-FUNCTION
264
266
PROVIDES: GUI OPEN_FILE(IN FILENAME:STRING):
ATTRIBUTES: GUIWINDOW="MICRO?UFT WORD"
GUIKEYS="OO1-:
END COMPONENT

FIG. 14

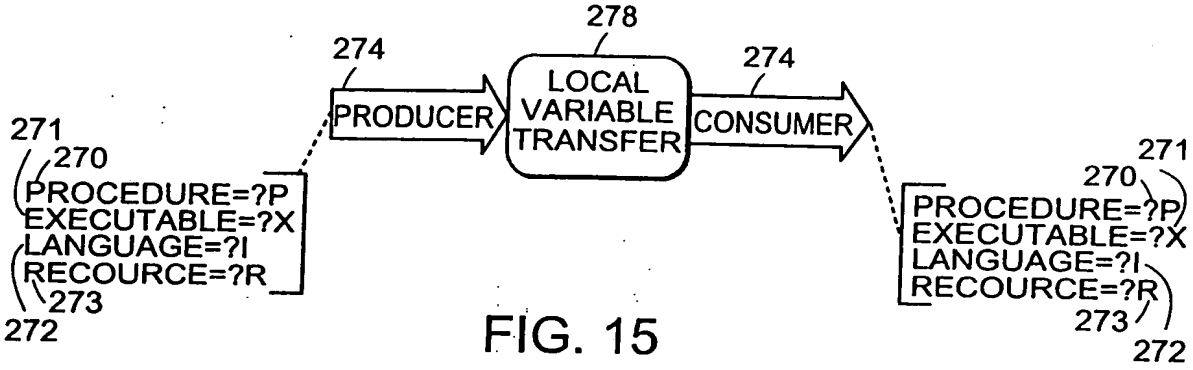


FIG. 15

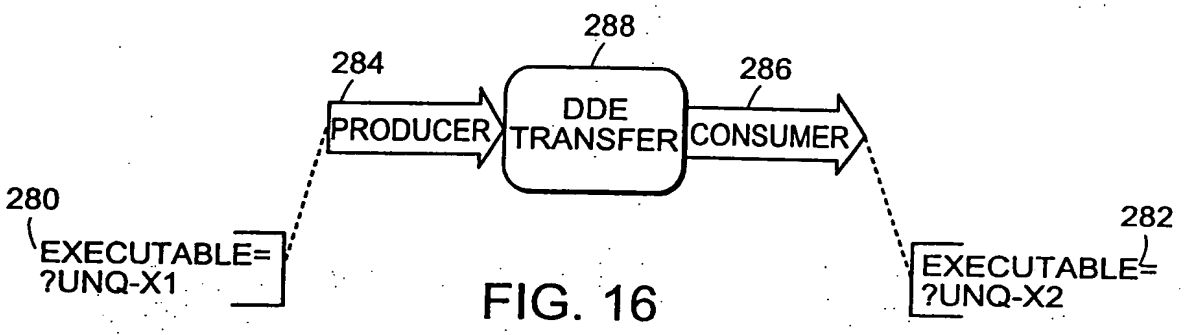


FIG. 16

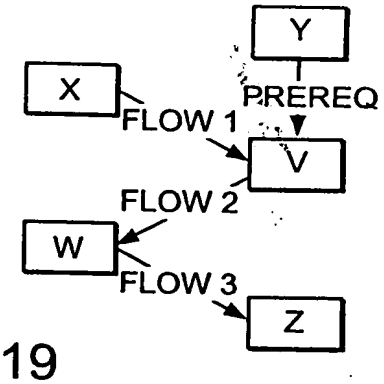
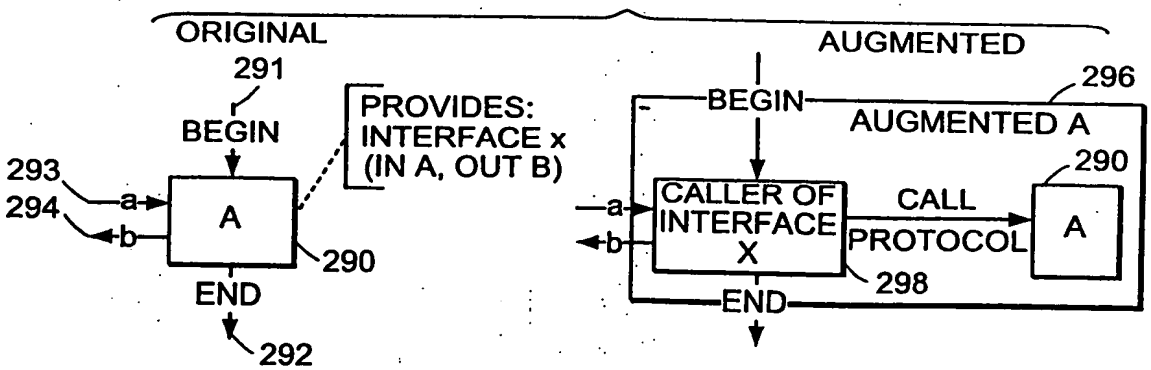


FIG. 18

FIG. 19



20040803.084120007

```
Check_Compatibility( aport, dport )
--              aport = activity port
--              dport = dependency port
-- Returns:     SUCCESS if ports can be legally connected, FAILURE otherwise
-- Uses:        Match_Values( va, vd )

460 - If one port is composite and the other is atomic then return FAILURE.

462 - If both ports are composite then recursively match subports.

464 - If both ports are atomic then
        If aport is same as or a specialization of dport then
            For each attribute defined at both ports (including inherited attributes)
                If both ends have a value then call Match_Values
            472 else If one end refers to a variable then
                    If variable has a value then call Match_Values
                    else    Set variable and its equivalence class to value at other end
            470 -           Return SUCCESS.
                else If both ends refer to variables
                    If one or both variables have values then do as above.
                    If no variable has a value then
                474 -       Unify both variables into an equivalence class
                            Return SUCCESS.
466 -   else return FAILURE.

468:
Match_Values( va, vd )
--              va = value of attribute at activity side
--              vd = value of attribute at dependency side
-- Returns:     SUCCESS if values match, FAILURE otherwise.

 If values are identical then return SUCCESS.
 If values are pointers to language elements then

476 -   If va is a specialization of vd*
            then return SUCCESS.
        *Exception: when comparing resources of consumer ports the opposite specialization relationship
must hold.
 Return FAILURE
```

FIG. 17

20040804-0040210

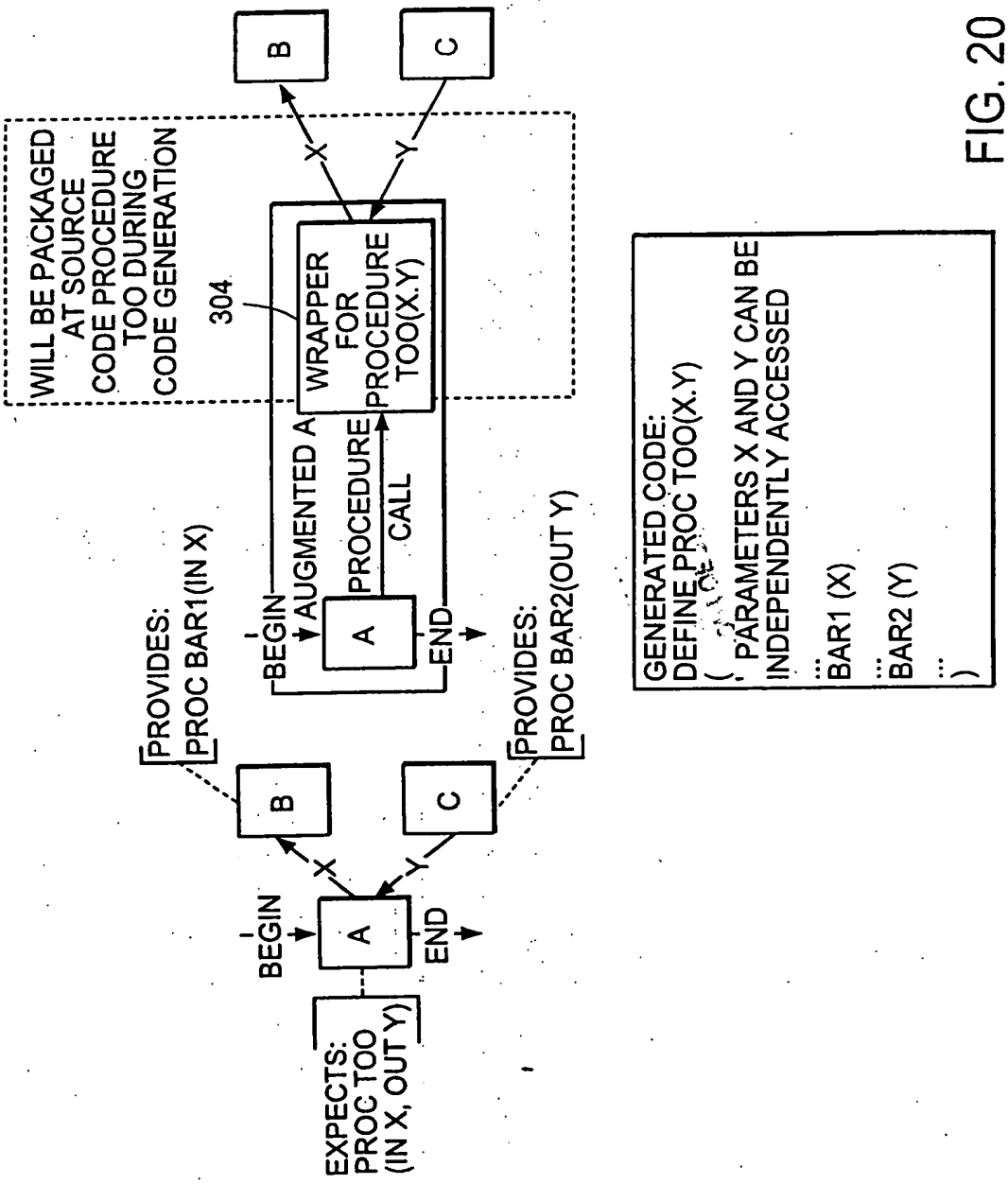

WILL BE PACKAGED AT SOURCE CODE PROCEDURE TOO DURING CODE GENERATION
304
WRAPPER FOR PROCEDURE TOO(X,Y)
AUGMENTED A
BEGIN
A
PROCEDURE CALL
END
B
C
X
Y
PROVIDES: PROC BAR1(IN X)
PROVIDES: PROC BAR2(OUT Y)
EXPECTS: PROC TOO (IN X, OUT Y)
GENERATED CODE:
DEFINE PROC TOO(X,Y)
(' PARAMETERS X AND Y CAN BE INDEPENDENTLY ACCESSED
...
BAR1 (X)
...
BAR2 (Y)
...
)

FIG. 20

2010h0" 03h2007

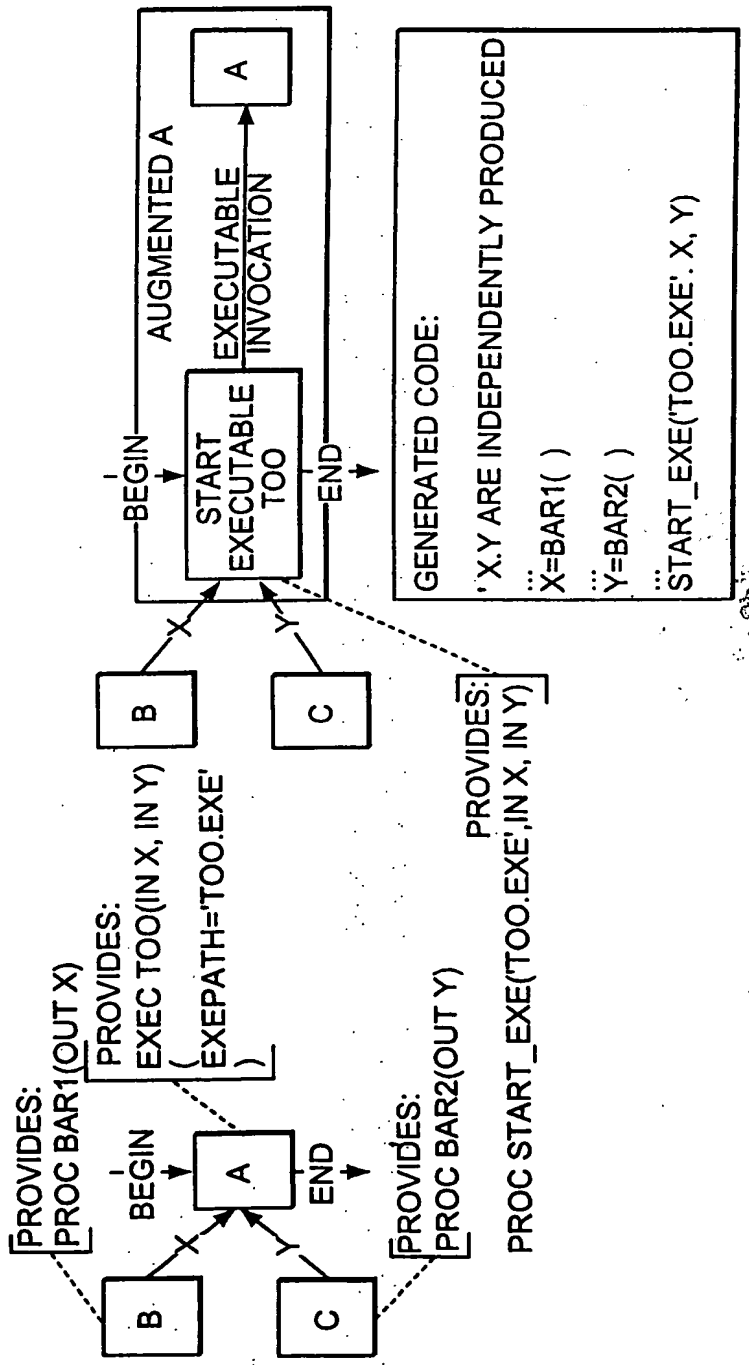


FIG. 21

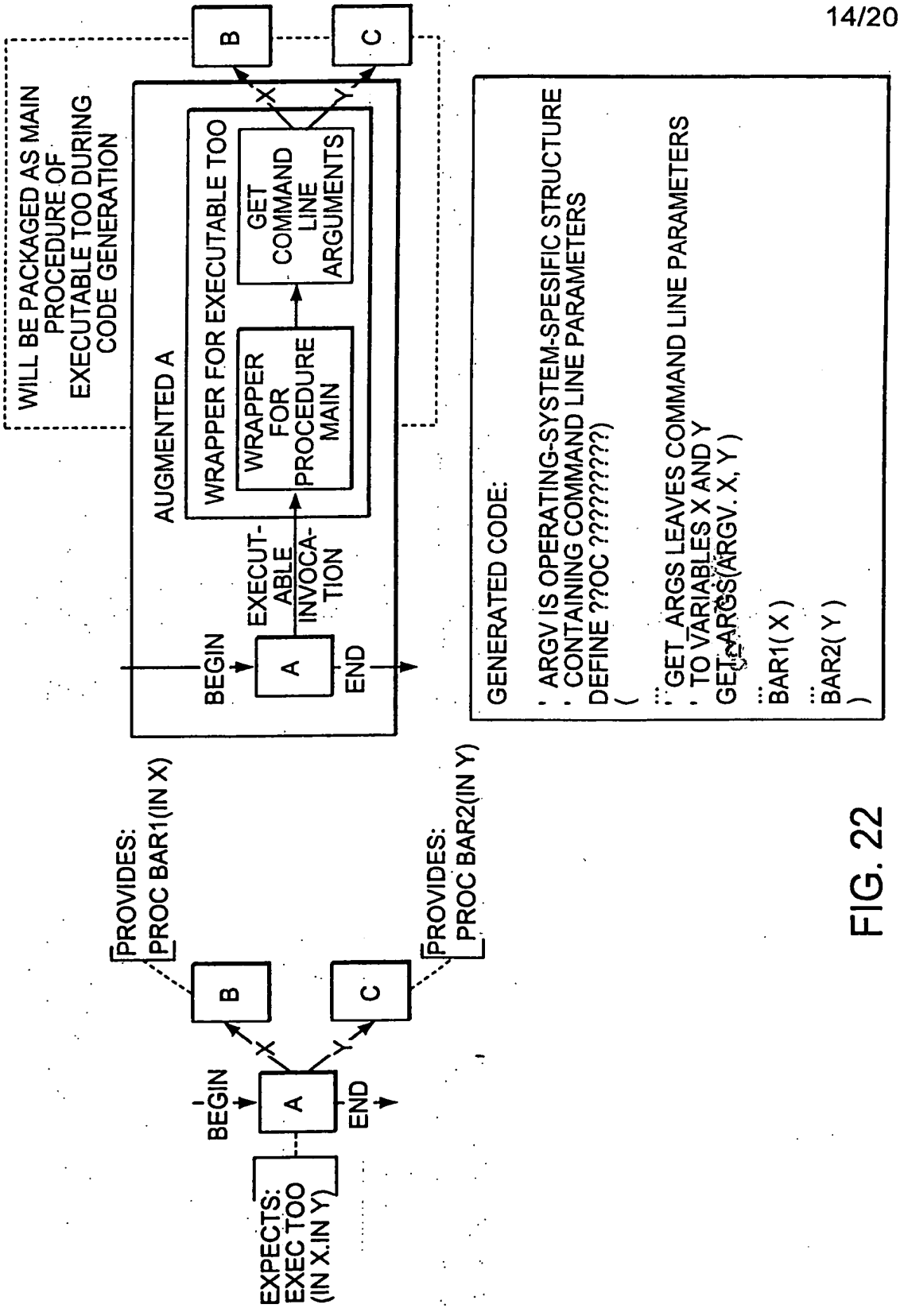

WILL BE PACKAGED AS MAIN PROCEDURE OF EXECUTABLE TOO DURING CODE GENERATION
AUGMENTED A
WRAPPER FOR EXECUTABLE TOO
WRAPPER FOR PROCEDURE MAIN
GET COMMAND LINE ARGUMENTS
EXECUT-ABLE INVOCA-TION
BEGIN
A
END
B
C
X
Y
GENERATED CODE:
' ARGV IS OPERATING-SYSTEM-SPESIFIC STRUCTURE
' CONTAINING COMMAND LINE PARAMETERS
DEFINE ??OC ????????
(
;;;
' GET_ARGS LEAVES COMMAND LINE PARAMETERS
' TO VARIABLES X AND Y
GET_ARGS(ARGV. X, Y)
;;;
BAR1(X)
;;;
BAR2(Y)
)
PROVIDES: PROC BAR1(IN X)
PROVIDES: PROC BAR2(IN Y)
EXPECTS: EXEC TOO (IN X.IN Y)

FIG. 22

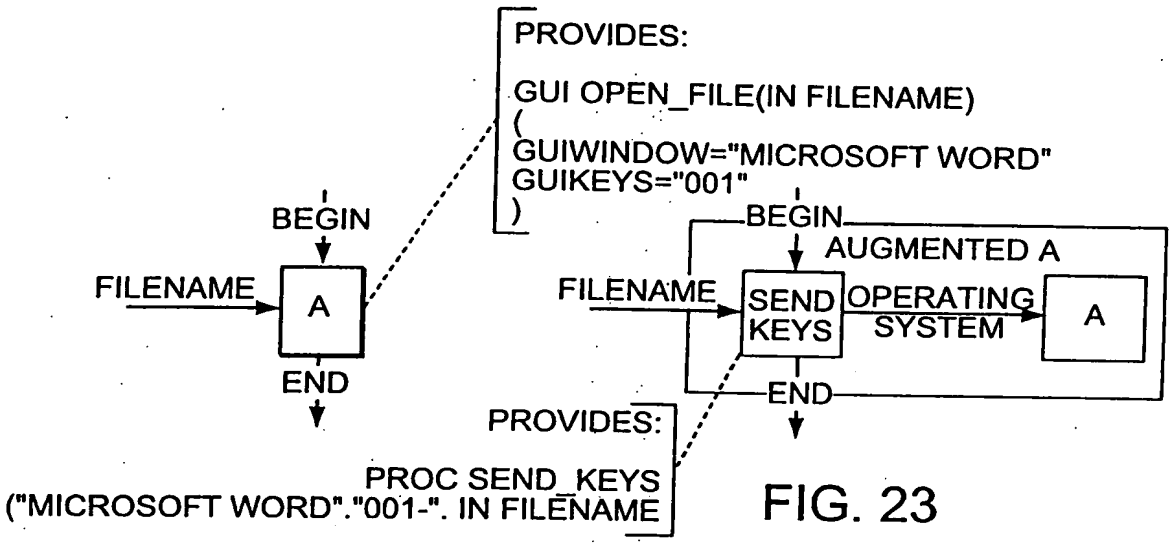


FIG. 23

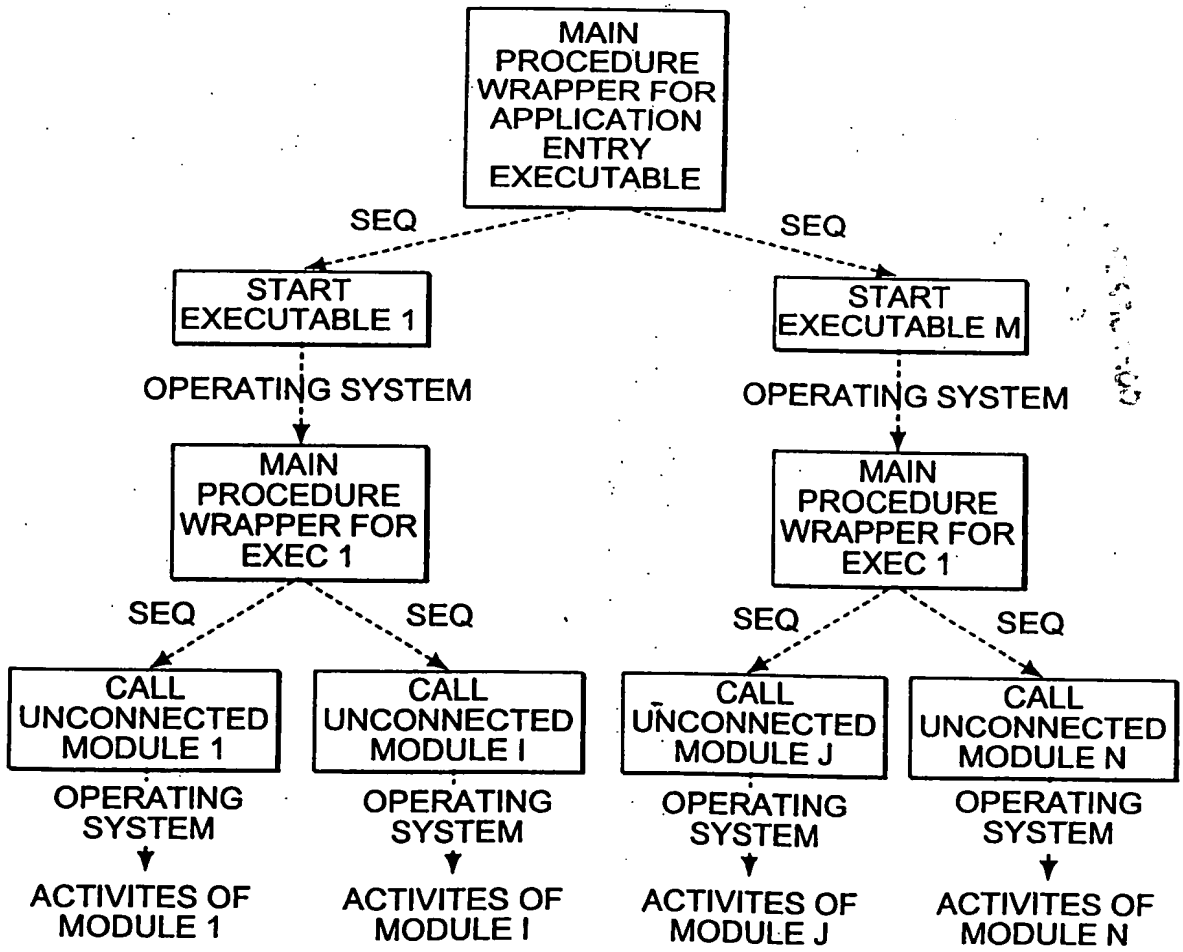


FIG. 24

20040108+08120005

Input: A diagram consisting of activities and dependencies
Output: A set of executable files implementing the target application

330. Decouple interface dependencies
332. Specialize generic design elements
334. Connect all modules to control
336. Generate executable code

FIG. 25

338 - Recursively scan all activities in the application graph.
For every activity associated with a code-level component,
340 - Scan all provided and expected interface definitions of the associated component.
For every provided interface,
342 - Get the interface kind.
If a caller activity has been defined for that interface kind,
344 - Check for "perfect match" special cases
If no "perfect match" interface is found at the other end,
346 - Replace the original primitive activity with a composite pattern that includes a caller activity.
For every expected interface,
348 - Get the interface kind.
If a wrapper activity has been defined for that interface kind,
350 - Check for "perfect match" special cases
If no "perfect match" interface is found at the other end,
352 - Replace the original primitive activity with a composite pattern that contains a wrapper activity.

FIG. 26

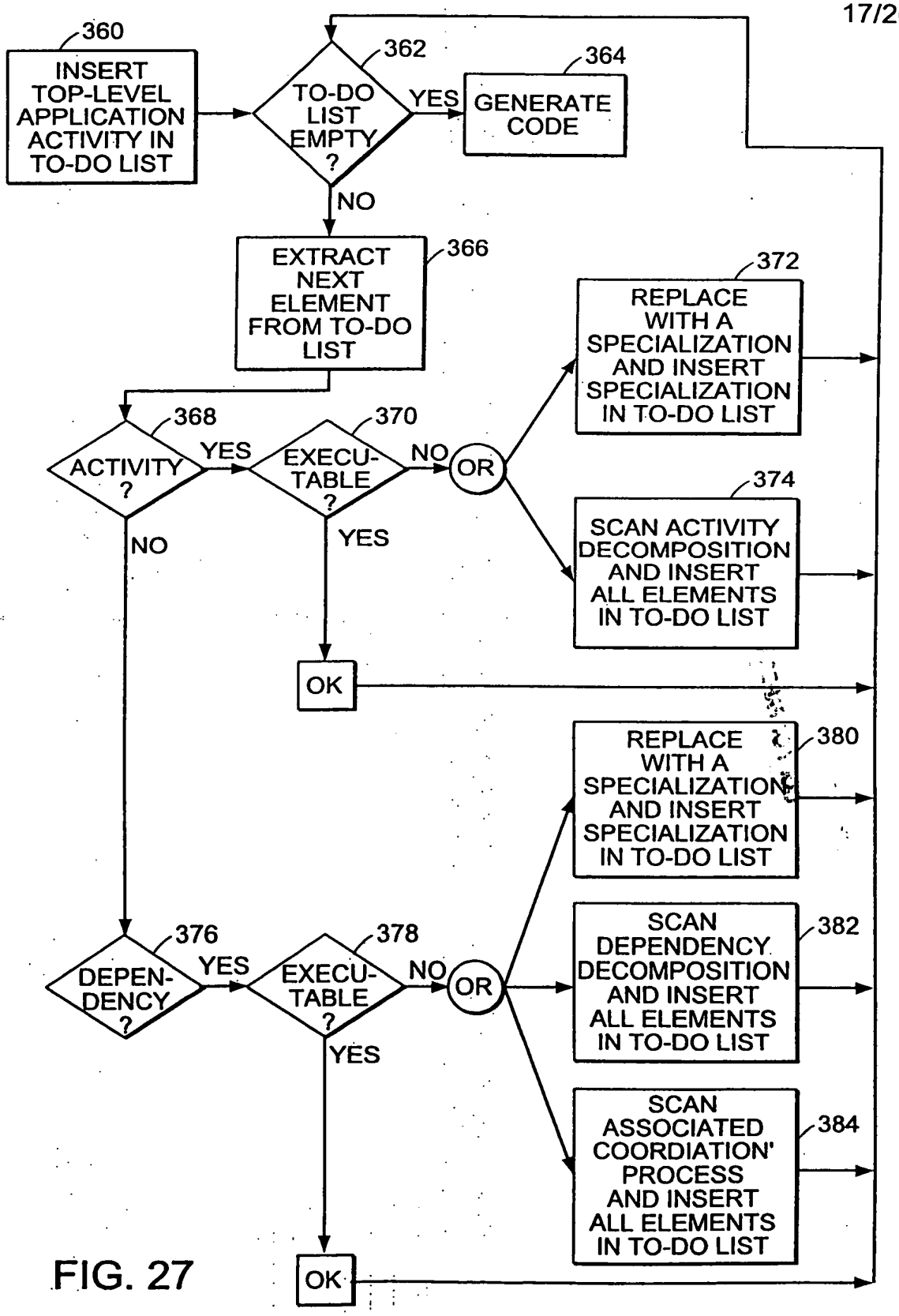


FIG. 27

20040408.0845

390: Scan graph and build a *to-do list* containing,
- all generic atomic activities (i.e. atomic activities not associated with a code-level component)
- all unmanaged dependencies

Repeat the following two operations until to-do list becomes empty,

392:a. Extract the next generic atomic activity.
For each executable specialization of that activity stored in the design repository,
396 Apply the compatibility checking algorithm
398 If at least two matching executable specializations are found,
Ask user to select between them.
Otherwise,
Repeat while user input is invalid:
400 Ask user to provide a specialization for the activity.
Check validity of user-supplied activity (if must pass the compatibility checking algorithm and either be *atomic and executable* or *composite*
402 Permanently store new activity in the repository.
Replace generic activity with selected or user-supplied specialization.
406 Apply Stage 1 of the algorithm to the replacing activity.
If replacing activity is composite and generic,
408 Scan activity decomposition and add all generic atomic activities and unmanaged dependencies found to the to-do list

394:b. Extract the next unmanaged dependency
For each coordination process* associated with a specialization of that dependency stored in the design repository,
410 Apply the compatibility checking algorithm.
If at least two matching coordination processes are found,
412 Ask the user to select among them.
Otherwise,
Repeat while user input is invalid:
414 Ask user to provide a compatible coordination process.
416 Check validity of user-supplied process (it must pass the compatibility checking algorithm and either be *atomic and executable* or *composite*)
418 Permanently store new process in the repository.
420 Manage dependency with the selected or user-supplied coordination process.
422 Apply Stage 1 (Fig. 26) of the algorithm to the managing coordination process.
If managing coordination process is composite,
424 Scan process decomposition and add all generic atomic activities and unmanaged dependencies found to the to-do list.

* The term coordination process here also includes atomic software connectors associated with executable dependencies.

FIG. 28

20040804.0804.20040804

430. Scan the application graph and find all source modules that are not connected to a source of control.
432. Introduce a set of *packaging* executable components, one per host machine and per language for which unconnected source modules exist.
434. Package calls to unconnected source modules inside the main program of the packaging executable corresponding to the host machine and language of each module.
436. Scan the application graph and find all executable programs that are not connected to a source of control.
438. Introduce an *application entry* executable component into the system.
440. Package invocation statements for all unconnected executables inside the main program of the application entry component.

FIG. 29

442 Scan graph and divide into *sequential block subgraphs**.
For each subgraph,
444 Topologically order activities according to their sequentialization interdependencies.
446 Generate a call statement for each activity.
448 Generate a local variable declaration for each local variable coordination process.
450 Generate appropriate headers and footers for the enclosing sequential block.
452 Save resulting sequential block code into a file.

For each target executable:
454 Collect all source and object files of the executable**.
456 Compile files and place resulting executable into target application directory.

* Sets of activiteis that will be packaged in the same sequential code block.

**The files are (1) all source and object files referenced in the component descriptions of all activities to be included in the executable, and (2) all coordination code source files generated by step 442 for the target executable.

FIG. 30

20040803

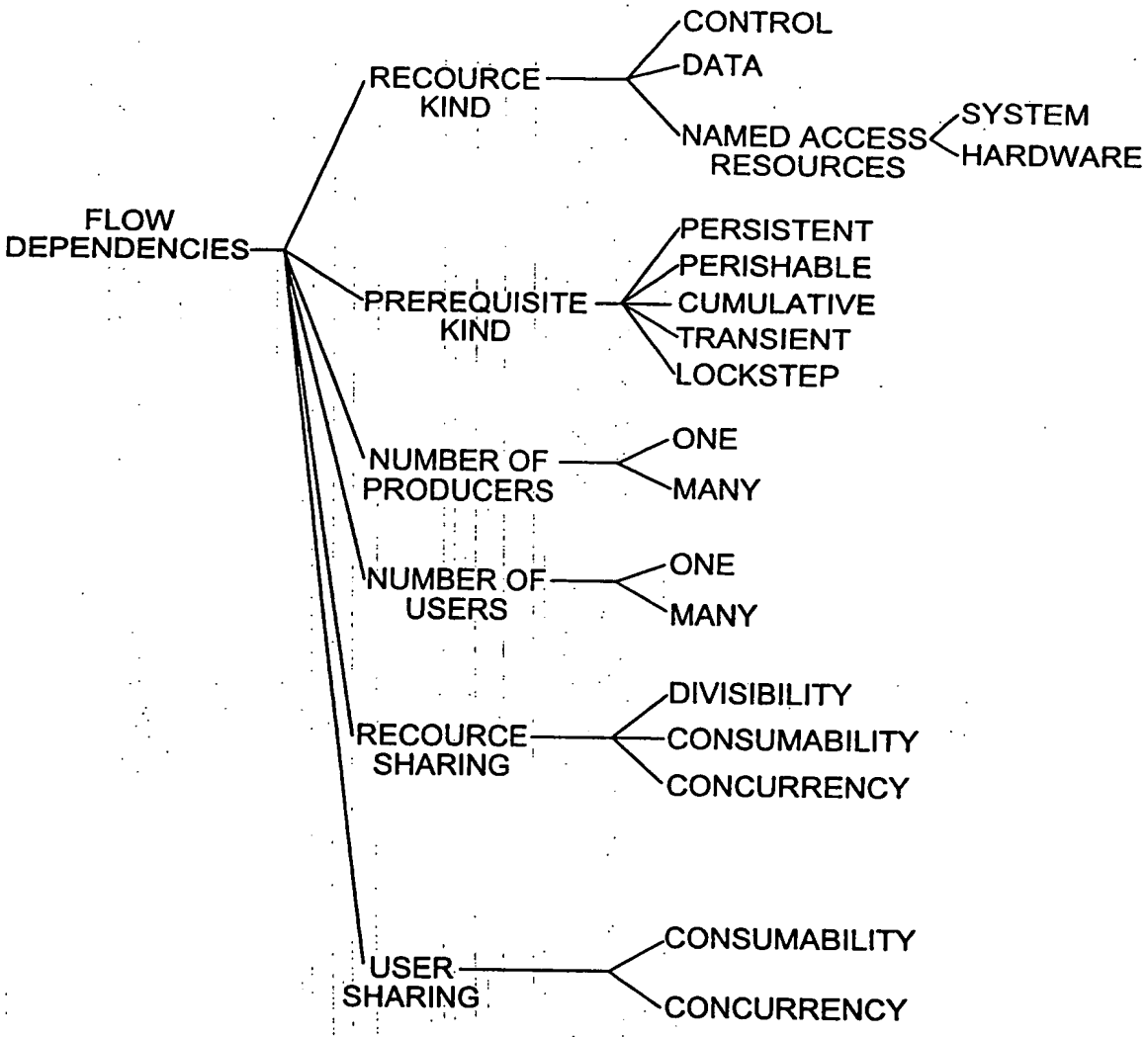


FIG. 31

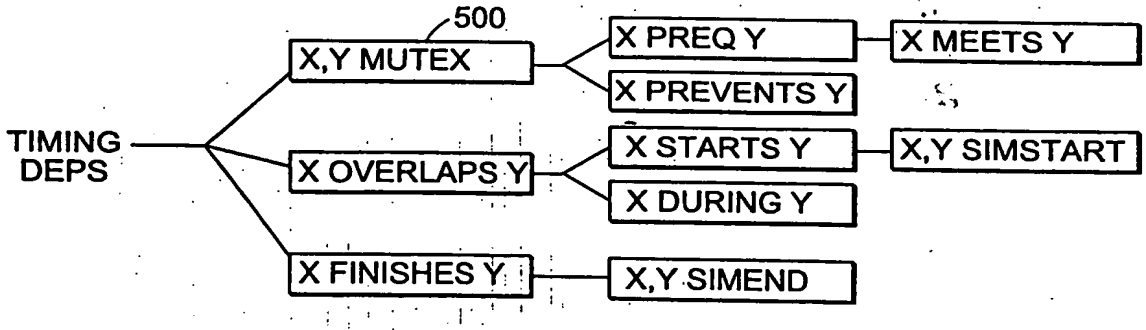


FIG. 32

20150410 054200 3